De-anonymization of Dynamic Online Social Networks via Persistent Structures

Tianchong Gao*, Feng Li*
*Indiana University-Purdue University Indianapolis, Indianapolis, IN, U.S.A.
tgao@iupui.edu, fengli@iupui.edu

***Abstract*—Service providers of Online Social Networks (OSNs) periodically publish anonymized OSN data, which creates an opportunity for adversaries to de-anonymize the data and identify target users. Most commonly, these adversaries use de-anonymization mechanisms that focus on static graphs. Some mechanisms separate dynamic OSN data into slices of static graphs, in order to apply a traditional de-anonymization attack. However, these mechanisms do not account for the evolution of OSNs, which limits their attack performance.**

In this paper, we provide a novel angle, persistent homology, to capture the evolution of OSNs. Persistent homology barcodes show the birth time and death time of holes, i.e., polygons, in OSN graphs. After extracting the evolution of holes, we apply a two-phase de-anonymization attack. First, holes are mapped together according to the similarity of birth/death time. Second, already mapped holes are converted into super nodes and we view them as seed nodes. We then grow the mapping based on these seed nodes. Our de-anonymization mechanism is extremely compatible to the adversaries who suffer latency in relationship collection, which is very similar to real-world cases.

***Index terms*—Dynamic online social networks; de-anonymization; persistent homology.**

I. Introduction

With the development of online social networks (OSNs), many service providers, e.g., Facebook and Twitter, release their OSN data to researchers and third-parties. This data can be used to recommend friendships, feed advertisements, and analyze the behaviors of communities. However, attackers can also use the OSN data to capture more sensitive information of the target users.

The attack upon the OSN data, i.e., the de-anonymization of published OSN data, mainly focuses on identify the target users in the released graphs. The adversaries can build an auxiliary graph with their background knowledge. Then the task of finding the target users is transformed into a graph mapping problem. If the adversaries successfully map the nodes from their auxiliary graph into the nodes in the released graph, they can take advantage of the information in the released OSN, e.g., the relationships in the graph and the salary amounts in the profile.

Some existing de-anonymization mechanisms examine both the structure similarity and the attribute similarity of nodes from two graphs [7, 11].. These mechanisms choose seeds that have high similarities. They first map the seeds and then map their neighbors. However, the structure change, which is introduced by both errors in adversaries' background knowledge and the noise injected by anonymization mechanisms, greatly affects the performance of existing de-anonymization attacks [4].

Moreover, OSN data is time variant although existing de-anonymization attacks mainly focus on the static graphs. For example, Facebook periodically releases their up-to-date OSN data, and the adversary sequentially add his/her new knowledge to the auxiliary graph. Researchers also designed de-anonymization attacks to dynamic OSNs. However, most of these mechanisms use the same methods that are used in de-anonymization attacks upon static data. Here, a time-series graph is considered as a combination of pieces of graphs [2]. Then the method to de-anonymize dynamic graphs is merely to sequentially de-anonymize static graphs, and then combine the results together. These mechanisms cannot use time to conduce de-anonymization. Therefore, the de-anonymization attacks upon dynamic OSN data may face the same problems that are faced when trying to de-anonymize static OSN data.

As mentioned above, the main challenge in de-anonymizing dynamic OSNs is that how to extract the time variant information and how to employ this kind of information in de-anonymization. Unlike other kinds of data, e.g., tuples of records in database, OSN data has a more complex data structure. Adding/deleting edges should not be viewed as simply changing pieces of edge records, without analyzing the changing impact to topology information. In this paper, we apply persistent homology to give the topology description of the time-series graphs. In particular, persistent homology can extract the persistency of hole structures under different dimensions and under different time slices [3, 15]. Persistent homology barcodes show the birth time and death time of the holes. We examine the similarities between holes in two time-series graphs, instead of individually considering the similarities between nodes in each piece of graph. If two holes match with each other, we use the nodes on the holes as seeds to further grow the node mapping, until two time-series graph are mapped.

The main contributions of our work are as follows:

- We probabilistically model the ability of adversaries to get the true relationships, which is realistic in real-world cases.
- We apply persistent homology to capture the persistent structures in dynamic graphs, to extract the time variant topology information.
- We introduce a seed-and-grow algorithm to map nodes in dynamic graphs, considering both structural similarity and attribute similarity.
- We experimentally analyze the scheme with real-world datasets and show the proposed scheme obtaining high accuracy in de-anonymization.

This is the author's manuscript of the article published in final edited form as:
Gao, T., & Li, F. (2019). De-Anonymization of Dynamic Online Social Networks via Persistent Structures. ICC 2019 - 2019 IEEE International Conference on Communications (ICC), 1–6. https://doi.org/10.1109/ICC.2019.8761563

Meaning	Symbol
Node	u, v, P-Y
Latency of attacker's information collection	α
Error of attacker's information collection	β
Structure similarity	M_s
Attribute similarity	M_a
Neighbor degree list	NDL
Cost function	C
Weight	w
Mapped neighbors	$N1$
Un-mapped neighbors	$N0$

TABLE I: Symbol table

II. Threat model

For reference, some important symbols used in this paper are given in Table I.

In this paper, an OSN is modeled by a time-series graph $G = (V, E)$. E is the set of vertices, where each node is a user in the OSN. E is the set of edges, where each edge is a relationship between two users. We measure both the activate/deactivate behaviors of friendships, like follow/unfollow, add/remove a friend, retweet/ignore contents. These behaviors are modeled as edge addition/deletion in the time-series graph.

Being a time-series graph, G also equals to a sequence of static graphs, i.e., $G = \{G_1, G_2, ..., G_n\}$, where G_i is the graph in i-th epoch. There are two main graphs considered in our scheme, G^A and G^B. G^A is published by the OSN service provider. G^B is built by the attacker with his/her background knowledge. The procedure of the published graph G^A and the background knowledge graph G^B. Our de-anonymization attack can be summarized as a procedure of mapping nodes between G^A and G^B. After successfully mapping the nodes, the attacker can utilize the information in G^A to enhance his/her background knowledge, i.e., collect more private information about the users.

However, the mapping task has two major challenges. First, the service provider does not directly release the graph. The service provider not only removes the identities in each static graph, but also perturb the graph before releasing. In the original graph, because new links may form and old links may drop, the self mapping of G^A is as hard as the de-anonymization work between two static graphs.

Second, the attacker cannot have perfect background knowledge graph, which is reflected in G^B. In this paper, we model the shortfall of the attacker's background knowledge by two factors: latency and unknown. Latency means that for some relationships forming/dropping in epoch i in the original OSN, the attacker knows this relationship is forming/dropping in epoch j, where $j \geqslant i$. Unknown means the attacker always have wrong information about some specific relationships. Particularly, if an edge is added/deleted in G_i^A, G_j^B adds/deletes that corresponding edge with probability p.

$$p = 1 - \beta \cdot \exp^{-\alpha \cdot (j-i)}, \quad j \geqslant i. \tag{1}$$

Two scaling parameters, α and β, represent the similarity between G^A and G^B. $\alpha \in [0, \infty)$ shows the latency of the attacker to collect the information. If $\alpha = \infty$, the attacker instantly collects the edge forming/dropping information. β shows the error of the attacker's collected information. If $\beta = 0$, there are no unknown relationships to the attacker. Having an edge forming in epoch i_1 and dropping in i_2, the existing probability of this edge in epoch j is

$$p = \begin{cases} \beta, & j \in [0, i_1), \\ 1 - \beta \cdot \exp^{-\alpha \cdot (j-i_1)}, & j \in [i_1, i_2), \\ \beta \cdot \left(\exp^{-\alpha \cdot (j-i_2)} - \exp^{-\alpha \cdot (j-i_1)}\right), & j \in [i_2, \infty). \end{cases} \tag{2}$$

Algorithm 1 Self mapping

Input: Two neighboring slices of G^A, G_i^A and G_{i+1}^A.
Output: A node mapping between users in G_i^A and G_{i+1}^A.

```
Set G^L = G_i^A, G^R = G_{i+1}^A,
In G^L and G^R, both select k users with highest degree values as seeds,
for each pair of seeds u and v do
    Compute similarity score M^1 = M_s^1 + θ M_a^1,
end for
Exhaustively search mapping results to get max Σ_{u,v} M^1,
for each pair of seeds u and v do
    Compute similarity score M^2 = M_s^2 + θ M_a^1,
end for
loop
    Pick a node u with the BFS algorithm,
    Find the best match nodes v with max M^2.
end loop
```

III. De-anonymization attack scheme

Given two dynamic graphs G^A and G^B, our goal is to map the nodes between G^A and G^B. The general idea of the scheme is to use the persistent structures as seeds in mapping. Our scheme has the following steps:

1) Self mapping: Having $G^A = \{G_1^A, G_2^A, ..., G_n^A\}$, if the node identifiers are not retained in G^A, we map the nodes between the series G^A.
2) Persistent structure extracting and mapping: We extract the barcode and the corresponding structures of G^A and G^B. Then for each persistent structure in G^A, we try to find a similar structure in G^B.
3) Match growing: After getting the pairwise persistent structures between G^A and G^B, we use the persistent structures as seeds to grow node mapping.

A. Self mapping

In some cases, the time-series graph is anonymized and then published by the OSN provider. The adversaries cannot have coherent identities of users in each epoch. Hence, we need to first mapping nodes of graphs in different epochs in G^A.

The process of self mapping a dynamic graph is similar to the process de-anonymization process with static graphs. Particularly, we sequentially de-anonymize the graphs G_1^A with G_2^A, G_2^A with G_3^A, and so on. For each mapping, we perform a two-stage de-anonymization attack in Algorithm 2 [10]. The general idea of Algorithm 2 is that the pop stars are more difficult to hide during anonymization than common users. Moreover, there is low probability that these popular users are newly added/deleted in OSNs.

In the two graphs G^L and G^R, e.g., G_1^A and G_2^A in the dynamic graph, we first choose k nodes with the highest

TABLE II: Example of calculating the optimal matching cost C

NDL(u)	5	3	0
NDL(v)	4	2	1
Costs	-1	-1	+1

degree. To each pair of nodes, which chosen from G^L and one chosen from G^R, we calculate a similarity score M. The similarity score considers both the structure similarity M_s and the attribute similarity M_a.

M_s is based on the cost C of optimally matching two neighbor degree lists (NDLs) [12]. Specifically, for the chosen node in G^L, we first get its neighbor list and the corresponding degrees. Then we construct the NDL of that node and compare it with the NDL of the node in G^R. Finally, we calculate M_s.

$$M_s(u,v) = -C(\text{NDL}(u), \text{NDL}(v)), \quad u \in G^L,\ v \in G^R. \tag{3}$$

See Table 1 for an illustration of calculating costs to determine M_s. If, for example, the node u in G^L has two neighbors with degrees 3 and 5, and the node v in G^R has three neighbors with degrees 1, 4, and 2, then we can use the method shown in Table II to calculate the cost C. First, we transform the two NDLs into decreasing order and add 0 to make them having the same length. Second, we calculate the differences between the two bits in the same position. Third, we get the total cost C, which is the sum of the absolute values of all costs. In this example, C equals 3 and $M_s = -3$.

M_a is the similarity of two users' attributes. We use the Jaccard index to measure the two sets of attributes $A(u)$ and $A(v)$.

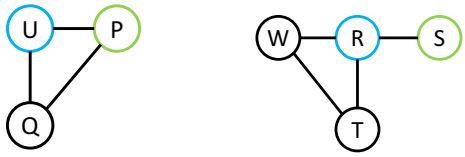
$$M_a^1(u,v) = \frac{A(u) \cap A(v)}{A(u) \cup A(v)}. \tag{4}$$

Then we have $M^1 = M_s^1 + \theta M_a^1$, where θ is a scaling parameter to balance structure similarity and attribute similarity. After getting the similarity score for top-k users, we assign a bipartite matching process to obtain the maximum sum of scores. Because the k users only occupy a very small part of the graph, we can exhaustively search all matching pairs to get the optimal result. We set a threshold M_t in order to prevent mismatching, in the case that some seeds in G^L are not seeds in G^R. If the similarity score $M < M_t$, that pair is eliminated.

After matching the seeds of G^L and G^R, we further grow the user mapping based on these seeds. Similarly, each pair of nodes, except the seeds, have a similarity score $M^2 = M_s^2 + \theta M_a^2$. While the attribute similarity is the same as the one in Equation 4, the structure similarity score considers the current and potential matching pairs. Specifically, each node has two sets: $N1$ shows the mapped neighbors, and $N0$ shows the unmapped neighbors. The similarity score is given by both the mapped neighbors (with Jaccard index) and unmapped neighbors (with elements count).

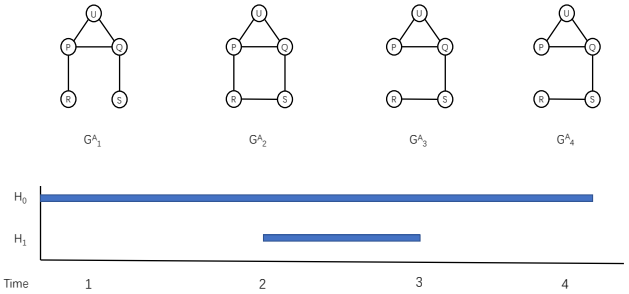
$$M_s^2(u,v) = \frac{N1(u) \cap N1(v)}{N1(u) \cup N1(v)} - \frac{||N0(u)| - |N0(v)||}{\max(N0(u), (N0(v)))}. \tag{5}$$

In the example of Fig. 1, we have two pairs of already mapped nodes, $U - R$ and $P - S$. Consider the structure similarity between Q and T: we have $N1(Q) = \{U, P\}$, $N0(Q) = \emptyset$, $N1(T) = \{R\}$, $N0(T) = \{W\}$. Then the similarity score is $M_s^2(Q,T) = \frac{1}{2} - \frac{1}{1} = -0.5$.

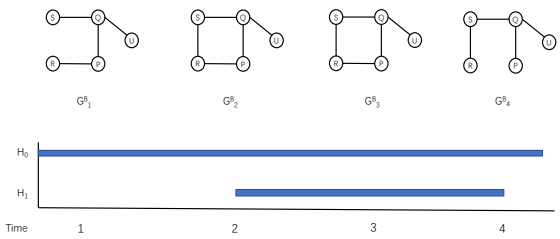


(a) Graph G^L (b) Graph G^R

Fig. 1: Example of growing mapping



(a) Time-series graph G^A and its barcode



(b) Time-series graph G^B and its barcode

Fig. 2: Example of hole mapping

After getting the similarity scores for all pairs of users, we need to do another round of bipartite matching to get the optimal mapping result. However, the searching space is almost all the users (except the seeds), which is much larger than seed mapping. Hence, we implement a heuristic searching method based on the breadth-first-search (BFS) algorithm. We set the seeds as the first layer of the tree to do BFS. The intuition of our algorithm is that the users neighboring the seeds should map with each other first to grow the mapping result.

B. Persistent structure extracting and mapping

Persistent homology is a utility metric that summarizes the graph in multi-scales. Persistent homology is presented in the form of barcodes [3]. In OSN graphs, a H_1 hole is a polygon with at least 4 sides. A polygon with at least 4 sides implies that all nodes on the polygon have at least one node which is not directly connected, while the triangles have all nodes pair-

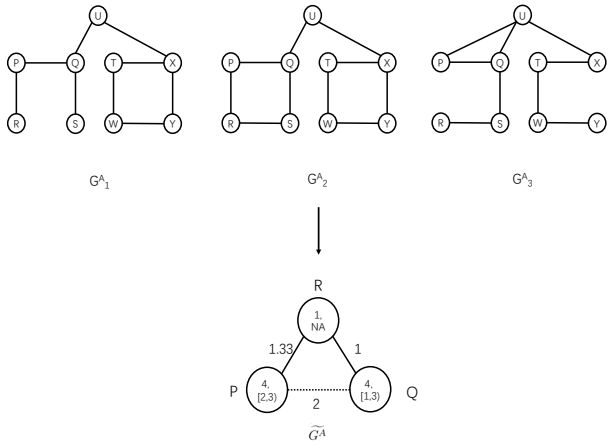


Fig. 3: Example of converted graph $\widetilde{G^A}$

wisely connected. Persistent homology barcodes can capture the birth time and death time of these polygons.

Fig. 2 gives a simple example of hole mapping. G^A has a hole from G_2^A to G_3^A. Then its barcode has an H_1 bar $[1, 2)$. G^B has a hole from G_2^B to G_4^B. Then its barcode has an H_1 bar $[1, 3)$. If the two holes match each other, the nodes along the hole, $\{P, Q, R, S\}$, are successfully mapped. Also, the node outside the hole, e.g., U in the example, can find its mapping with the help of the seed nodes on the hole. In the example, the barcode is not perfectly matched. However, this scenario also occurs in real-world cases because the adversaries' relationship building or breaking information may have errors. With the help of persistent homology, we can extract the similarity over several continuous time periods.

Each persistent structure has three important features: the number of nodes involved in the structure, birth time, and death time. When we map the persistent structures, each structure is combined into a super node. We assign a weight w on each edge. There are two kinds of edges in the converted graph: (1) edges between two super nodes, and (2) edges between a simple node and a super/simple node. Hence the weights also have two meanings. On the first kind of edge, the weight shows the distance between two super nodes. On the second kind of edge, the weight shows the connectivity of the two nodes. Finally, we have the converted graph $\widetilde{G^A}$ based on the original graph G^A.

For example, Fig. 3 shows a converted graph $\widetilde{G^A}$ and its original time-series graph G^A. In the converted graph $\widetilde{G^A}$, the node P with label '$4, [2, 3)$' means that it is a super-node containing the persistent structure with 4 nodes, and the persistent structure exists from time 2 to time 3. The node R with label '$1, \mathrm{NA}$' means that it is a simple node. The edge P-Q has a weight $w = 2$ because the two persistent structures have the minimum distance 2 in all static graphs of G^A. The edge P-R has a weight $w = 1.33$, which equals to the total number of edges, which is (1+1+2), divided by the number of time slots, which is 3. This weight shows that on the average 1.33 edges exist between the two nodes in all time slots.

After getting the converted graph, we map the nodes according to this graph. Specifically, we first map super nodes in this graph as seeds, then we grow the map. When we map super nodes, we temporarily discard all simple nodes and related edges, e.g., solid edges in $\widetilde{G^A}$ in Fig. 3.

Algorithm 2 Seed and grow mapping with weight

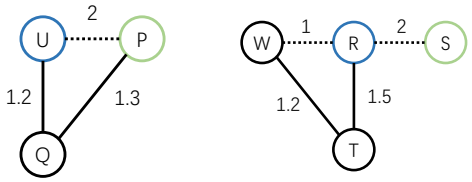
Input: Two weighted graph $\widetilde{G^A}$ and $\widetilde{G^B}$
Output: A node mapping result between users in $\widetilde{G^A}$ and $\widetilde{G^B}$

```
——————————Seed mapping——————————
Set G^L = G̃^A, G^R = G̃^B,
In G^L and G^R, both select k users with highest degree values as seeds,
for each pair of seeds u and v do
    Compute similarity score M^3
end for
for each pair of seeds u and v with the highest M^3 do
    Exhaustively search mapping results to get max M^4,
end for
——————————Growing——————————
for each pair of simple nodes u and v do
    Compute similarity score M^5 = M_s^5 + θ M_a^1
end for
loop
    Pick a node u with the BFS algorithm,
    Find the best match nodes v with max M^5
end loop
```

The mapping of super nodes has the following steps:

1) We divide the super nodes into groups according to the first number on their label (which indicates the number of nodes involved).
2) For nodes in the same group, we calculate the dissimilarity, which equals the differences among birth times and death times. We have
$$M^3 = \begin{cases} -\infty, & \text{if } birth_{G^A} > birth_{G_B}, \\ & \text{or } death_{G^A} > death_{G_B}. \\ -(\Delta birth + \Delta death), & \text{otherwise.} \end{cases} \quad (6)$$
Then each possible mapping pair has a similarity score.
3) Begin with a mapped pair with the highest M^3, we iteratively try to map other nodes to get the maximum M^4, so we have
$$M^4 = \sum_{all\ pairs} M^3 \cdot \exp^{-w}. \quad (7)$$
where w is the distance weight.
4) Then we change the initial mapping of step 3 to another pair, choosing the pair with the second highest M^3, as we get the maximum M^4. We repeat this step and record the best initial mapping and the following mapping.

Because the persistent structures with different sizes have a low probability to of showing the same group of users, we divide the super nodes into groups in step 1. Step 2 ensures that two persistent structures have a probability of mapping together when they have similar birth times and death times, but the existence periods are not required to be exactly the same. Hence, even if some relationship information is not expediently collected by the adversary, our mapping algorithm still has the chance to map the persistent structures together.



(a) Graph G^L (b) Graph G^R

Fig. 4: Example of growing mapping with weighted edges

Note that if the adversary has incorrect information of edge addition or deletion, which happens in the true case of an OSN, we do not map the persistent structures together.

The intuition behind step 3 is that we need to take the distance of persistent structures into consideration. The farther the distance, the less impact the structure has upon central mapping. However, the best M^4 calculated in step 3 is only meaningful to the initial mapping. It may be locally optimal result. Hence, in step 4 we iteratively change the initial mapping pair and get the best mapping result. In real cases, the initial mapping pairs may be the persistent structures with the same sizes and existence periods. So we need to test all possibilities when initial mapping has $M^3 = 0$. Although the final result may not be the globally optimal result, different initial pairs help our heuristic algorithm get better performance without an exhaustive search.

C. Match growing

In the match-growing process, we need to map the simple nodes with the help of persistent structures. Moreover, we need to differentiate the nodes inside each persistent structure. First, the whole graph $\widetilde{G^A}$ should be recovered, which means the simple nodes, solid edges, and the connectivity weights are back.

Compared the match-growing process discussed in Section III-A, the match growing here needs to consider the weights, i.e., the connectivities, between the simple nodes and the super nodes. Each pair of simple nodes has a similarity score $M^5 = M_s^5 + \theta M_a^1$. The attribute similarity is the same as the one in Equation 4, and the structure similarity is given by

$$M_s^5(u,v) = -\frac{\sum_{N(u),N(v)} \Delta w}{\sum_{N(u)} w + \sum_{N(v)} w} - \frac{||N0(u)| - |N0(v)||}{\max(N0(u),(N0(v)))}. \quad (8)$$

The first item, based on Δw, shows the dissimilarity of weights between mapped seeds. The second item is the same as the one in Equation 5, which shows the unmapped neighbors.

IV. Experiment

In this section, we evaluate our de-anonymization algorithm with a real-world dataset, Facebook wall network [6, 13]. This dataset collects users' posts to other users' walls on Facebook from 2005 to 2009. The nodes of the network are Facebook users, and each edge means one post. Since users may write multiple posts on a single wall, the dataset collects each post and its timestamp. Fig. 5 shows the number of edges in each month.

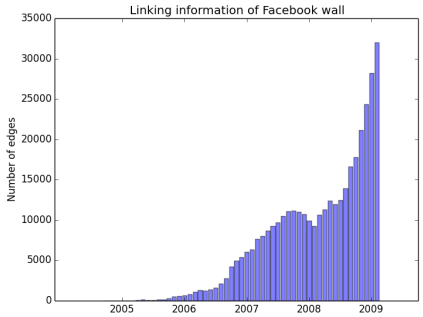


Fig. 5: Number of edges in each month

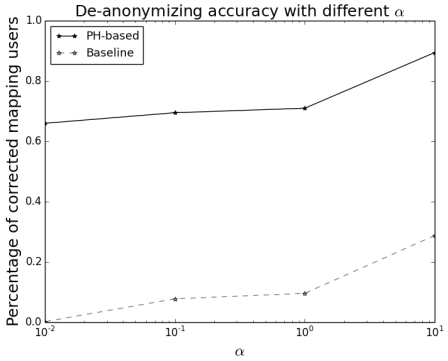


Fig. 6: Mapping accuracy with different α, $\beta = 0.2$

In our evaluation, we consider a post as a linking relationship between two users. And we combine time slices into time periods. If two users do not post anything in one time period, we consider their relationship to be broken in this time period. Here we set the length of time period to three months, which is a reasonable length to measure a relationship. Also, three-month period divides the whole time series graph into 17 time periods, which means the time series graph sequentially has 17 static OSN graphs. There are 46k users and 274k edges in the dataset among all time periods. We combine the whole dataset into a dynamic graph to capture the birth and death information of persistent structures.

For the purpose of comparison, we reimplement a baseline de-anonymization attack described in [2]. In the baseline approach, the mapping probabilities of node pairs are individually calculated in each graph slices. Then, Ding et al. combined these probabilities and calculated the product of these values to find the most similar nodes. In the presented results, the baseline approach and the proposed persistent homology based approach are marked as 'Baseline' and 'PH-based', respectively.

We examine the effectiveness of our algorithm when the attacker cannot have update information about relationships and his/her information has some error. In our model, we use the parameter α to represent the latency of attacker's knowledge, and we use β to represent the error in attacker's knowledge. First, we fix the value of β to 0.2, which means the attacker can only get 80% of correct information in the end, even without considering the latency. It is more likely to the real scenarios that the attacker has limited ability in

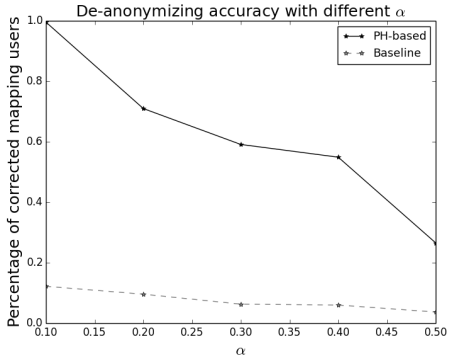


Fig. 7: Mapping accuracy with different β, $\alpha = 1$

collecting information. We get the highest mapping accuracy when $\alpha = 10$, and the mapping accuracy is 89.44%. When alpha equals 0.01, 0.1, 1, and 10, the mapping accuracy of the proposed method is 66.0%, 69.5%, 71.0%, and 89.4%, respectively. The mapping accuracy of the baseline approach is 0.2%, 7.8%, 9.6%, and 28.8%, respectively.

We can find that higher the α, higher the accuracy. Timely collect information helps the attacker successfully de-anonymize the users in our scheme. However, the latency of the adversaries' ability to capture information does not affect the mapping accuracy very much. Since most of the late edge addition/deletion in G^B will be corrected in the following time periods, our algorithm checks similarity between persistent structures in different time periods. Unlike other algorithms, which are required to precisely map edges in each time, our algorithm has the ability to capture the similarities among different time periods. Hence, our algorithm is robust to late edge addition/deletion.

Then, we also fix the latency parameter α to 1, and test our de-anonymization scheme with various amount of error information. Fig 7 shows the mapping accuracy with different β. We get the highest mapping accuracy when $\beta = 0.1$, and the mapping accuracy is 99.58%. When β equals 0.2, 0.3, 0.4, and 0.5, the mapping accuracy of the proposed method is 70.98%, 59.11%, 54.90%, and 26.56%, respectively. The mapping accuracy of the baseline approach is 12.2%, 9.6%, 6.3%, 6%, and 3.7%, respectively. We find that the error amount in adversaries' background knowledge largely impacts the de-anonymization ability of the adversaries. If the adversaries can capture the true information of edge addition/deletion, our algorithm is able to capture the persistent structures and de-anonymize the users.

V. Related research

Graph matching is widely used in OSN de-anonymization [9]. In graph matching, sometimes attackers choose high similarity nodes to help them applying a seed-based attack [1, 14]. These seed-based attacks huge success in de-anonymizing static graphs [5]. Existing seed-based attacks give us guidance about the seed and grow process, and considering structure similarity and attribute similarity together. However, when analyzing dynamic OSN de-anonymization, how to combine information from variant time slices data together to find the seeds becomes a big challenge.

Existing de-anonymization attack to dynamic OSNs naively combine slices of graphs. In [2], the overall probability of mapping two nodes is the product of mapping probabilities in all time-series graphs. In [8], the similarity of path building time is analyzed to map two nodes together. Although these attacks embed some temporal features in de-anonymization, there is not enough temporal information to describe the evolution of OSNs, especially when the OSN graphs are complex.

VI. Conclusion

In this paper, we propose a new de-anonymization algorithm to deal with the dynamic OSNs. We introduce the persistent structures to capture the edge addition/deletion among different time periods. Our algorithm can map two similar persistent structures without having the same edge building time. The evaluation result shows the effectiveness of our algorithm, especially when the adversaries have less incorrect information.

References

[1] Dalal Al-Azizy, David Millard, Iraklis Symeonidis, Kieron O-Hara, and Nigel Shadbolt. A literature survey and classifications on data deanonymisation. In *International Conference on Risks and Security of Internet and Systems*, pages 36–51. Springer, 2015.

[2] Xuan Ding, Lan Zhang, Zhiguo Wan, and Ming Gu. De-anonymizing dynamic social networks. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–6. IEEE, 2011.

[3] Robert Ghrist. Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

[4] Shouling Ji, Weiqing Li, Prateek Mittal, Xin Hu, and Raheem A Beyah. Secgraph: A uniform and open-source evaluation system for graph data anonymization and de-anonymization. In *USENIX Security Symposium*, pages 303–318, 2015.

[5] Shouling Ji, Weiqing Li, Shukun Yang, Prateek Mittal, and Raheem Beyah. On the relative de-anonymizability of graph data: Quantification and evaluation. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016.

[6] Jrme Kunegis. KONECT – The Koblenz Network Collection. In *Proc. Int. Conf. on World Wide Web Companion*, pages 1343–1350, 2013.

[7] Huaxin Li, Qingrong Chen, Haojin Zhu, Di Ma, Hong Wen, and Xuemin Sherman Shen. Privacy leakage via de-anonymization and aggregation in heterogeneous social networks. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[8] Jun Long, Lei Zhu, Zhan Yang, Chengyuan Zhang, and Xinpan Yuan. Temporal activity path based character correction in heterogeneous social networks via multimedia sources. *Advances in Multimedia*, 2018, 2018.

[9] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 173–187. IEEE, 2009.

[10] Wei Peng, Feng Li, Xukai Zou, and Jie Wu. A two-stage deanonymization attack against anonymized social networks. *IEEE Transactions on Computers*, 63(2):290–303, 2014.

[11] Jianwei Qian, Xiang-Yang Li, Chunhong Zhang, Linlin Chen, Taeho Jung, and Junze Han. Social network de-anonymization and privacy inference with knowledge graph model. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[12] Matthias Studer and Gilbert Ritschard. What matters in differences between life trajectories: A comparative review of sequence dissimilarity measures. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 179(2):481–511, 2016.

[13] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in Facebook. In *Proc. Workshop on Online Social Networks*, pages 37–42, 2009.

[14] Yazhe Wang and Baihua Zheng. Preserving privacy in social networks against connection fingerprint attacks. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 54–65. IEEE, 2015.

[15] Xiaojin Zhu. Persistent homology: An introduction and a new text representation for natural language processing.